Authenticated File Broadcast Protocol

Simão Reis[1], André Zúquete[2], Carlos Faneca[1], and José Vieira[2]

[1] IEETA / University of Aveiro, {simao.paulo,carlos.faneca}@ua.pt
[2] DETI / IEETA / IT / University of Aveiro, {andre.zuquete,jnvieira}@ua.pt

Abstract. The File Broadcast Protocol (FBP) was developed as a part of the DETIboot system. DETIboot allows a host to broadcast an operating system image through an 802.11 wireless network to an arbitrary number of receivers. Receivers can load the image and immediately boot a Linux live session. The initial version of FBP had no security mechanisms. In this paper we present an authentication protocol developed for FBP that ensures a correct file distribution from the intended source to the receivers. The performance evaluations have shown that, with the best operational configuration tested, the file download time is increased by less than 5%.

1 Introduction

The DETIboot system is a solution that was designed and developed to quickly install a temporary, live Linux image in an arbitrarily large number of computers [1,2,3]. It uses a wireless 802.11 network (WiFi), operating in ad hoc mode, and broadcast communication to send the Linux image to nearby clients. These load the image and immediately boot a Linux live session, which can disappear without leaving a trace after a power down. This system has potential applications in both academic and enterprise environments. Currently we are working on a security ecosystem for DETIboot in order to allow its use in exams using students' personal laptops and an hardened Linux image.

The DETIboot system uses a broadcast file distribution protocol (FBP). The first version of FBP had no security mechanisms, which is not advised for ensuring a correct distribution of the intended Linux image among all receivers. In this paper we propose a broadcast authentication protocol for FBP, which enables FBP receivers to make a correct download from an intended FBP server.

Our authentication protocol kept the basic behaviour of FBP. The authentication is performed with extra messages (authenticators) interleaved at unpredictable places within the original FBP frame transmission flow. These authenticators enable a set of future frames to be authenticated by the receivers. The option for sending an authenticator before its target frames was taken for preventing receivers from accumulating frames that may never be authenticated.

Authenticators use well-known technology: SHA-1 frame digests, signed with an RSA private key. For the envisioned exploitation scenarios we use high-performance RSA setups (1024 bit modulus, small public exponents) for increasing performance without compromising security. The setup of an FBP session

is driven by parameters derived from the public key of the FBP server, which guaranties that receivers cannot be fooled by networks deployed by attackers.

For evaluating the performance of our proposal, we did measurements in multiple operational scenarios, considering different reception conditions, different timings for sending authenticators and the presence of an attacker. Without attacks, the overhead in the total download time for a more aggressive transmission of authenticators, the one with better results, was below 5%.

This paper is structured as follows. In Section 2 we present the FBP concepts that are fundamental to understand the options we took for adding authentication, as well as its security weaknesses. In Sections 3 and 4 we present our FBP authentication protocol and some implementation details. In Section 5 we present the performance evaluation of FBP with authentication. In Section 6 we present some related work in the area of broadcast authentication. Finally, in Section 7 we present our conclusions.

2 File Broadcast Protocol

In this section the File Broadcast Protocol (FBP) is described in order to clarify the scenario that needs to be protected.

FBP [1,2,3] uses Fountain Codes [4,5] to broadcast a file. Fountain Codes create a sequence of codewords that can be generated from a given set of source symbols such that those symbols can be recovered from any subset of the codewords of size equal to, or only slightly larger than, the number of source symbols.

FBP starts by slicing the file to be transmitted into a set of equally-sized segments (source symbols). Pseudo-random XOR combinations of those segments are then calculated, yielding Fountain Code codewords. FBP repetitively and indefinitely broadcast codewords from a server to multiple clients. The transmitter encodes file segments into codewords and the clients decode those codewords, obtaining the original file segments.

FBP clients may enter at any time in the broadcast session, they do not need to be present at the begging of the FBP transmission. Furthermore, they are tolerant to packet losses since, in theory, it does not matter the exact set of codewords one needs to receive (all codewords are equality good to get the original symbols). After a given threshold of received codeword it should be highly probably to complete the decoding and get all the original symbols.

FBP is a network protocol (layer 3 of the OSI model), identified through the Ethernet code 0x1986 in 802.11 frames. An 802.11 codeword frame conveys the codeword itself and the indexes of all symbols used to generate it. The indexes are not transmitted directly, they are derived by receivers from parameters included in the frame: total number of symbols (K), degree of the codeword (number of symbols used in its generation), and a random seed (cf. Figure 1). The degree and the seed, together with a universal pseudo-random generator, are used to generate the index set of the symbols contained in the codeword. A codeword frame also includes a codeword index (a sequence number) for performance evaluation purposes. We used it also for defining authentication windows.

2.1 Security vulnerabilities / attacker model

FBP uses a wireless medium, through 802.11 ad-hoc networks, to broadcast codewords of a boot image from a source to many destination laptops. Thus, an attacker can try to impersonate a legitimate source in order to provide its own boot image. Alternatively, the attacker may provide only a few codewords that would act as a Trojan Horse, i.e., could change the final behavior of the boot image while keeping most of its functionality unchanged. This is not easy, but certainly not impossible.

Besides those attacks, where the attacker could attempt to control the downloaded boot image, an attacker can use Denial of Service (DoS) attacks. In this case, it can (i) repeat previously sent codewords or (ii) prevent legitimate codeword transmissions or receptions.

In the first case, which is a typical replay attack, it would increase the receivers' memory with useless codewords, but not ruin the codeword decoding process (repeated codewords can, in fact, happen). In any case, it is advised to discard repeated codewords if we are able to detect such situation.

In the second case, involving interference with a legitimate transmission and reception, there is no definitive solution, because jamming or abusive 802.11 medium occupancy is always possible. Nevertheless, we can make an attacker's task harder by forcing it to interfere continuously with the legitimate transmitter. In fact, since FBP codewords do not need to be strictly ordered and can be lost, as long as a receiver is able to get codewords, even at a lower rate, it will continuously evolve towards a complete codeword decoding. The only way to avoid this is by preventing receivers to get any codewords at all.

Another type of attack involves the name resolution used in ad hoc networks. These networks, formally referred as Independent Basic Service Sets (IBSS), can be identified by names, which are assigned to 48-bit values (BSS IDentifiers, BSSID). Usually, the binding between a network name and a BSSID is made by any node that attempts a network name resolution within the neighbors and gets no answer. Thus, it is perfectly possible to have a legitimate FBP transmitter and an attacker with the same network name bound to different BSSID values. In this case, FBP receivers should not resolve network names to BSSID values, because they may get the attacker's BSSID, thus entering its network thereafter. In such case, the attacker could impersonate the legitimate FBP transmitter, providing its own boot image, or remain silent, this way deploying a DoS black hole attack. Since the name resolution is a basic 802.11 feature, which cannot be changed or protected in any way, the obvious solution is to force the BSSID of FBP receivers to a value known to be in use by the legitimate FBP transmitter.

Finally, assuming that FBP needs some mechanism to enable receivers to check the validity of the codewords they receive, this mechanism should be designed in a way that does not allow an attacker to interfere with it with a minimum effort. Otherwise, it would be easy for an attacker to force receivers to discard all legitimate codewords. The autonomous authentication of each and every codeword could be a solution, but the overhead costs, both in terms of data transmission and CPU processing, could as well be excessive. On the other

hand, the transmission of a few, critical authentication control frames, which could be used to authenticate many codewords, cannot be predictable (i.e., nobody should be able to guess their transmission slot). Otherwise, an attacker could simply jam those control frames to interfere with, an completely ruin, the entire codeword reception process.

2.2 Authentication requirements and alternatives

As referred in [6], the solutions to reliable, point-to-point packet communications do no scale well to broadcast environments. In point-to-point is usual that receivers request retransmission of the data in case of failure. FBP solves this issue for broadcast communication by using Fountain Codes, which do not require feedback. Furthermore, authentication in point-to-point communications can be achieved with a pure symmetrical solution, such as a Message Authentication Code (MAC). Both parties share a common secret key, and when a message with a correct MAC arrives the receiver is assured that the correct transmitter generated it. However, in a broadcast environment a MAC is not safe. Every party knows the MAC key, therefore anyone could impersonate the genuine source and assume the broadcast transmission. The obvious approach is the use of an asymmetric mechanism, such as a digital signature. These have the asymmetrical authentication property required by FBP: each source generates signatures with its private key and the receivers can verify the signatures with the public key of the intended source.

An FBP receiver needs to know something about a legitimate FBP transmitter to authenticate it, or the codewords it sends. However, an FBP receiver should process codewords immediately upon their reception, to maximize the decoding CPU cycles between the reception of consecutive frames. Consequently, it is advised to either (i) authenticate each codeword independently from the others or (ii) to transmit a multi-codeword authentication frame prior to a transmit the respective codewords.

The natural solution for the first option is to include in each codeword a signature, produced by the FBP transmitter and that could be verified by each an every receiver. However, ordinary signatures, such as the ones made with RSA, can take a relevant space in the codeword frame. For a typical codeword with nearly 1500 bytes, an RSA signature using a 1024-bit modulus would add 128 more bytes to a codeword. This means about 9% more data to transmit per codeword. Since to overall decoding time is a function of the time it takes to receive a minimum number of codewords, with this authentication strategy the overall decoding time would increase by no less than 9%. This is not dramatic, but we thought we could get a better solution, and we did.

The second option is to transmit authentication frames with authentication material for the frame itself and for checking a set of codewords following it. A simple strategy for implementing this authentication policy would be to include in a authentication frame a set of references and digests of future codewords, all signed by the FBP transmitter. With a 1500-byte frame and an 128-byte RSA signature we have room for about 60 20-byte SHA-1 digests. Without frame

losses, this strategy has an overhead lower than 2% relatively to the data transmitted, because we only need to transmit an authenticator (and the correponding public key to validate it) before a batch of 60 codewords. However, with frame losses the overhead is higher, because upon the loss of an authenticator the receiver would have to discard all codewords until getting the next authenticator.

Regarding attacks, the second option is potentially weaker than the first against DoS attacks. In fact, if an attacker could predict the instant when authenticators are transmitted, then it could jam the network during such transmission and, with a minimum effort, could prevent the validation of all received codewords. However, this weakness can be mitigated by adding some randomness to the instants when authenticators are transmitted. For instance, the transmitter can insert a variable number of codewords between authenticators, ranging from 1 and up to the maximum of digests present in the previous authenticator. With this strategy, an attacker could never anticipate the transmission of an authenticator, therefore selective jamming would not be possible any more.

3 Authenticated File Broadcast Protocol

This section presents the authentication extension developed for the FBP protocol. In this extension we used the last solution presented in the previous section: special authentication packets, interleaved from time to time with codewords, which authenticate a fixed number of following codewords.

3.1 Design assumptions and options

Our authentication protocol was conceived for an operational environment where a new, fresh asymmetric key pair can be create and used in a time-limited file download session. For instance, it can be use to download a particular live Linux distribution in the beginning of a class, possibly taking no more then a few minutes. Or we can use a daily key pair for on-demand distribution of live Linux distributions for the computers of an organization (e.g. a demonstration distribution for all laptops being presented in shelfs of a computer store).

In both cases, we take the two following assumptions: (i) key pairs can be changed frequently, on a per-session basis, and do not need to stay stable for a long time; and (ii) the receivers can get, from a reliable source, some elements that allow them to verify if a public key is the correct one they should use. In this last case, we did not consider any automatic validation strategies, such as public key certificates or certification chains, but rather some human-driven mechanisms, such as the validation of the equality between digests.

Taking into consideration the first assumption, there is no need to use very long asymmetric keys; we chose 1024-bit RSA keys. Furthermore, we used the smallest Fermat prime (3), as the public exponent, which reduces to the minimum the computation overhead in the receivers without bringing known security vulnerabilities. We have chosen SHA-1 as the algorithm to compute the digest of each codeword. Currently it has no known vulnerabilities and the digests are not excessively long.

3.2 Key distribution and validation

Each authenticator carries the modulus of the public key of its generator, as well as a signature produced by the corresponding private key. When a receiver starts, it waits for an authenticator, checks its signature, presents a digest of the public key to the user and waits for an accept/reject decision. This decision must be taken upon checking, by some means, if the digest is the expected one. For instance, in a classroom, the teacher controlling the source machine can get the same digest and write it in the board.

For segregating communications involved in different FBP sessions, the overall key-related setup is slightly more complex:

1. The sender initiates the FBP server, this generates a fresh key pair for the transmission session. Then, it computes (and displays) a digest from its public modulus and uses part of that digest to compute (and display) the BSSID of its ad hoc network. This BSSID can or cannot be already in use, that is irrelevant for FBP.
2. The receiver initiates the FBP client with the BSSID being used by the intended server, in order to enter its ad hoc network.
3. The FBP client waits for an authenticator, which it will use to present the digest of its public modulus. If the user approves its value, the modulus is recorded for checking future authenticators.
4. The FBP client waits for a valid (properly authenticated) codeword for extracting the download operational parameters – number of symbols K and size of each symbol/codeword. Once having this, the decoding process can start (using this and the following valid codewords).

Ethernet packets have a payload of 1500 bytes. Subtracting the size of the public RSA modulus (128 bytes), the size of the corresponding signature (128 bytes), the remaining space can, at the maximum, accommodate 62 20-byte codeword digests. We decided to use only 60, leaving some space in the authenticator for some extra fields that could be necessary.

Each authenticator can authenticate 60 consecutive codewords. Considering that authenticators are equal in size to codewords (the difference is small), at least $\frac{1}{61}$ ($\sim$1.6%) of all transmitted bytes will be exclusively used for authentication. By increasing the transmission frequency of authenticators we increase accordingly such overhead. This can be a low price to pay when transmission losses increase, affecting the number of received authenticators. We will address this issue in Section 5.

3.3 Authenticator generation

Before sending a set of $1 \leq N \leq 60$ previously generated codewords, the FBP server creates an authenticator to protect the next $60 \geq N$ codewords $(C_x, \cdots, C_{x+59})$. For each codeword C_i, with $i \in [x, x+59]$, a digest $d_i = h(C_i)$ is calculated and inserted into the authenticator. The authenticator also carries the index $x + 59$ of the codeword used to compute the last digest (d_{x+59}), an

codeword index	(4 bytes)
K	(4 bytes)
seed	(4 bytes)
degree	(4 bytes)
codeword	(1484 bytes)

last codeword index: $x + 59$	(4 bytes)
0	(4 bytes)
session identifier	(4 bytes)
SHA-1 digests: $d_x, d_{x+1}, \cdots, d_{x+59}$	(1200 bytes)
RSA signature & public key modulus	(256 bytes)

Fig. 1. FBP frame payloads for codewords (left) and authenticators for transmitting before codeword index x (right)

RSA signature of the transmitter over this index and all the digests and the signer public key modulus (see Figure 1).

Upon checking the validity of an authenticator, an FBP client saves all its digests to validate future codewords. For instance, if the last codeword index in the authenticator is 2000, only the 60 codewords with an index between 1941 and 2000 can be checked and possibly accepted by the client. If in the meanwhile another valid authenticator is received, this validation information is updated accordingly for authenticating the following codewords.

Figure 1 shows the complete physical mapping of an authenticator's fields. The second field, corresponding in terms of location to a codeword's field K, is always 0. Since K is never 0 in codeword frames, this field can be used by clients to distinguish codewords from authenticators. The session identifier is used to efficiently discard authenticators belonging to a different download session (sharing, by a very unlikely coincidence, the same 802.11 channel and the same BSSID value). This identifier is randomly chosen by an FBP server and adopted by a receiver upon the acceptance of an authenticator that carries the public modulus that will be used to authenticate the traffic.

3.4 Replay attacks against clients

To prevent replay attacks, authenticators with an outdated latest codeword index (lower than the one of the current authenticator being used) are discarded by clients without further validation.

The codewords' index is also checked to avoid replay attacks. The client saves the index of the last (valid) codeword and discards codewords with a previous or equal index without further validation. Note that we are working with a one-hop wireless network, where frames, in principle, do not get out of order.

4 Implementation details

4.1 Key generation and distribution

The FBP server was modified to generate an RSA key pair with the public exponent 3. Besides including the modulus of this key in all authenticators, the server outputs for its administrator an SHA-1 digest of the modulus and a 48-bit BSSID extracted from part of such digest. These two values need to be conveyed,

by the best suited means, to all the users running the FBP client and wishing to download a file from this server. The BSSID is used by the server to initiate the ad hoc network prior to start using it.

The FBP client was modified to accept a BSSID (formerly it was using a name-BSSID translation). Then it waits for an authenticator, displays the SHA-1 digest of its public key modulus, and prompts the user if that is the key to be used. The user has the options to (i) use it, (ii) do not use it once, (iii) do not use it forever, or (iv) input the SHA-1 digest or the desired modulus. Thus, flooding attacks exploring constant or always changing modulus on each authenticator can be overcome by either (i) choosing not to use a modulus forever or (ii) providing the digest of the correct modulus, respectively.

4.2 Production of authenticators

The server was implemented as a pipeline of 3 tasks: (i) produce the codewords, (ii) build the authenticators and (iii) send both the codewords and the authenticators. Most laptops nowadays are multiprocessor so these tasks can be assigned to an equal number of threads (Producer, Signer and Sender), each on its own CPU, in order to maximize the CPU usage. These tasks manage two circular buffers that are used for queueing codewords and authenticators (see Figure 2).

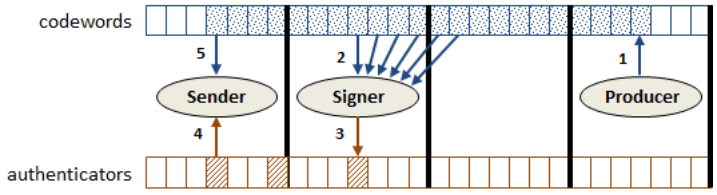

Fig. 2. Tasks, buffers and actions used to coordinate the production and transmission of codewords and authenticators

The Producer thread is the first to start. It produces as many codewords as possible (action 1 in Figure 2). As soon as there are enough codewords (at least 60) to fill an authenticator with their digests, the Signer thread starts. Upon having at least one authenticator, the Sender thread starts.

There are 4 synchronization points, implemented with semaphores: two between the Producer and the Signer; one between the Producer and the Sender and one between the Sender and the Producer. These are represented by the thick vertical lines in Figure 2. Preliminary tests indicated that synchronization points between any pair of buffer items would cause to much computational of overhead. So, these 4 synchronization points are done in blocks of 60 items. The Producer needs only one block; while inside it, it produces as many codewords as possible. The Signer needs two blocks, because each authenticator must always protect the next 60 codewords, and some may be in the next adjacent block. The Sender only needs one block. In total, 4 blocks of 60 items are needed.

As we can see in actions 2 and 3 in Figure 2, the Signer always saves the authenticator in the same index as the first codeword it authenticates. In actions 4 and 5 in Figure 2, the Sender at index i of a block b checks if in the authenticators' queue there is a authenticator for the same index. If there is, it sends it, and then it sends the codeword in the same position of the codewords' queue. Otherwise, it only sends the codeword in that position.

5 Performance Evaluation

The performance evaluation involved reaching two fundamental conclusions: (i) what is the overhead, under normal conditions (i.e., when not being attacked), of our FBP authentication and (ii) what is the preferable frequency for sending authenticators.

To reach these conclusions we have made a series of live measurements considering all possible scenarios combining the following parameters:

- A receiver near or far away from the transmitter. For the near case we experienced an average RSSI (Received Signal Strength Indicator) of -25 dBm, while for the far case we experienced an average RSSI of -75 dBm) (evaluated with Android mobile phones using the Wifi Analizer application).
- A variable number C of codewords between each authenticator, uniformly distributed in the intervals $[1, 30]$ and $[1, 60]$. The first interval leads to an higher frequency in the transmission of authenticators.
- The presence of another transmitter using the same wireless network (same 802.11 channel, same BSSID) and close to the receiver.

For transmissions without authentication we obtained the following indicators: (i) decoding elapsed time (**Time**); (ii) the number of codewords effectively used for reaching the complete file decoding (**Used**); (iii) the number of codewords received by the decoder but not effectively decoded (**Unused**); and (iv) the number of codewords lost in the transmission, due to physical transmission problems or overruns of reception buffers (**Lost**). This last value is computed from the indexes of the first and last codewords (F and L, respectively) received by the decoder and the total number of codewords received by the decoder (R):

$$\text{Lost} = L - F + 1 - R$$

Since $R = \text{Used} + \text{Unused}$, then

$$\text{Lost} = L - F + 1 - (\text{Used} + \text{Unused})$$

The (percentage of) codeword loss in the decoding process is given by

$$\text{Loss} = \frac{\text{Lost}}{R + \text{Lost}} = \frac{\text{Lost}}{\text{Used} + \text{Unused} + \text{Lost}}$$

For transmissions with authentication we obtained all the previous indicators plus the following: (i) the number of codewords from the correct source that

failed authentication (**Invalid**); and (ii) the number of codewords from other sources that also failed authentication (**Other**). To distinguish codewords from the correct or incorrect source we used the K of each codeword and all codeword sources broadcast files with a different K. With authentication the calculation of Lost is different, being given by

$$\text{Lost} = L - F + 1 - (\text{Used} + \text{Unused} + \text{Invalid})$$

and the (percentage of) codeword loss in the decoding process is calculated as

$$\text{Loss} = \frac{\text{Invalid} + \text{Lost}}{\text{Used} + \text{Unused} + \text{Invalid} + \text{Lost}}$$

In the measurements we used the following systems and data:

Legitimate transmitter: Toshiba Portégé 830-10R, with an Intel Core i7-2620M at 2.7 GHz, 8 GiB of RAM, with an external (USB) Thomson TG123g WiFi interface (with the TxOP option [7] for fast transmission[3]), running a 64-bit Linux Lubuntu. It was used to transmitted a file with $104,792,660$ bytes ($\sim$100 MiB, 70615 symbols, each with 1484 bytes).

Receiver: Asus K55VM-SX083V, with an Intel Core i5-3210M Dual Core at 2.5 GHz, 8 GiB of RAM, Atheros AR9485 WiFi interface, running a 64-bit Linux Lubuntu at runlevel 1 (single user administration mode).

Attacker: Asus F3SC-AP260C, with an Intel Core 2 Duo T5450 at 1.67 GHz, 1 GiB of RAM, running a 32-bit Linux Lubuntu.

Note that the attacker can be as powerful as intended, as it can be deployed with different machines. In our case, the attacker was made intensionally less powerful than the correct FBP source.

In all transmissions we used 802.11g broadcast at the maximum speed allowed by interface drivers. For the legitimate transmitter we could set that speed to 54 Mbit/s, the 802.11g maximum. When combined with the TxOP, the non-authenticated FBP can achieve a download performance of about 40 Mbit/s. Without such option, the maximum performance drops to about 25 Mbit/s.

Tables 1, 2 and 3 present the average and standard deviation values observed for the elements previously referred in the several scenarios considered. All values were computed after 10 experiments in the same exact circumstances.

5.1 Analysis of results

The results show a typical result of our coding policy: the number of codewords required for completing the file decoding is fairly stable in all cases. However, the time to get those codewords varies a lot depending on the scenario.

Regarding our first goal, compute the overhead introduced by the authentication in normal circumstances (when not being attacked), we see that the

[3] TxOP allows a transmitter to send batches of frames separated by the minimum possible time, a SIFS (Short Interframe Space).

Table 1. Results at the end of the file decoding without authentication

	Time (s)		Used		Unused		Loss (%)	
Distance	Avg	σ	Avg	σ	Avg	σ	Avg	σ
Near	20.8	0.9	72102.0	171.3	913.6	540.2	0.09	0.09
Far	70.6	6.4	72238.4	137.3	151.3	111.6	3.72	0.12

Table 2. Results at the end of the file decoding with authentication and no attackers

		Time (s)		Used		Unused		Invalid		Lost		Loss (%)	
Distance	C	Avg	σ	Avg	σ	Avg	σ	Avg	σ	Avg	σ	Avg	σ
Near	[1, 30]	21.5	0.1	72141.1	212.5	493.6	328.9	25.6	5.4	42.6	28.2	0.09	0.04
	[1, 60]	21.0	0.1	72212.7	236.4	591.3	343.0	161.2	53.5	437.7	87.4	0.82	0.12
Far	[1, 30]	74.0	14.5	71989.3	132.0	228.0	310.6	8107.3	2069.3	168894.7	46626.2	69.95	6.27
	[1, 60]	121.5	40.3	71943.1	156.3	147.2	270.7	30170.0	6624.8	314448.3	132831.6	81.34	4.94

Table 3. Results at the end of the file decoding with authentication and an attacker

		Time (s)		Used		Unused		Invalid		Lost		Loss (%)		Other	
Dist.	C	Avg	σ	Avg	σ	Avg	σ	Avg	σ	Avg	σ	Avg	σ	Avg	σ
Near	[1, 30]	35.5	0.4	71949.3	145.4	616.8	262.8	820.2	101.5	23006.3	681.2	24.7	0.6	20436.3	23.9
	[1, 60]	36.0	2.1	72155.2	409.9	548.2	281.4	7144.1	1329.5	20811.5	4384.4	27.6	4.0	20917.0	20.5
Far	[1, 30]	87.8	32.7	72061.8	363.9	238.3	330.3	9925.3	3728.8	209600.6	105077.0	72.3	9.3	2610.0	68.9
	[1, 60]	135.8	37.2	72033.8	174.8	38.8	68.2	35275.0	5758.2	358082.8	126205.5	83.7	4.1	3265.3	75.8

Table 4. Overheads in the decoding time and codeword losses due to authentication

Distance	C	Δ Time (%)	Δ Loss (%)
Near	1-30	3.5	0.5
	1-60	1.3	773.9
Far	1-30	4.8	1780.1
	1-60	72.2	2086.5

overhead is small. Table 4 shows the overheads due to the introduction of authentication. In terms of decoding time, the increment ranged from 3.5 to 4.8% when the frequency of authenticators is higher ($C \in [1, 30]$), and 1.3 to 72.2% when such frequency is lower ($C \in [1, 60]$). In terms of codeword losses, this value increased due to the discarding of invalid codewords. The increase was between 0.5 and 1780.1% when authenticators are more frequent, and between 773.9 and 2086.5% when authenticators are less frequent.

There is an apparently strange outcome, which is the fact that, despite a major increase of losses with authentication, the total decoding time does not increase on the same proportion. Notice, however, that if we have erasure rates (total losses) ϵ_n and ϵ_f for a nearby and far way transmissions, respectively, for reaching a threshold X of codewords in the decoder enabling it to complete the decoding we need to transmit N_n and N_f codewords, in each case, such that

$$X = N_n \times (1 - \epsilon_n)$$
$$X = N_f \times (1 - \epsilon_f)$$

which means that

$$N_f \times (1 - \epsilon_f) = N_n \times (1 - \epsilon_n) \Leftrightarrow \frac{N_f}{N_n} = \frac{1 - \epsilon_n}{1 - \epsilon_f}$$

Now, for $\epsilon_n = 0.0009$ and an $\epsilon_f = 0.6995$ (observed with high frequency authenticators, see Table 2), we get $\frac{N_f}{N_n} \approx 3.325$. Since there is some linear correlation between the decoding time T and the total number of transmitted codewords during the decoding (N), we can also anticipate that $\frac{T_f}{T_n}$ should yield a similar value, which it does: $\frac{74.0}{21.0} = 3.442$. This demonstration is also applicable to the results obtained for transmissions with less frequent authenticators.

Regarding our second goal, finding a preferable frequency for sending authenticators, the tests allow us to conclude that, except in one case, it is preferable to use a higher frequency ($C \in [1, 30]$) than a lower one ($C \in [1, 60]$). With an higher frequency the Time and Loss indicators, the ones that are relevant to evaluate the transmission efficiency, are usually lower than with a low frequency. The increase of Loss is partially due to the increase of the Invalid indicator, which grows when authenticators are transmitted less frequently.

The exception happens when the receiver is very close to the transmitter and there is not an attack. Besides being an hard-to-find scenario (not all receivers can be this close, specially when there are many or they are scattered along a classroom), the difference in the average decoding time is negligible ($\sim 2\%$) for deciding for a lower frequency.

When an attacker is present and competes for the transmission media, it will succeed in reducing the FBP performance. This is evident from the comparison of the results of Tables 2 and 3. However, such results show a curious behaviour: when the attacker and the victim are close to each other, an far away from the genuine source, the Other indicator drops when comparing with the scenario were all three hosts are near each other. This is probably due to transmission collisions between the genuine source and the attacker, which have difficulties in listening to each other traffic.

We have used SHA-1 both for computing the digests of codewords and the authenticators' signatures. Since SHA-1 is deprecated for digital signatures [8], we should probably use stronger digest functions, such as SHA256 or SHA512, for handling signatures. We did some experiments with SHA256 and the performance results were very similar to the ones observed with SHA-1, which means we can increase security without compromising performance.

6 Related Work

Regarding the authentication of Fountain Code transmissions, in [9] the authors developed a solution for authenticating Fountain Code codewords used in the distribution of a new image in multi-hop wireless sensor networks. Their solution is totally different from ours: they recode the original symbols to include digests of other symbols, forming an hash chain up to a new root symbol that needs to be transmitted authenticated and without Fountain Codes. The digests can only be recovered when original (recoded) symbols are recovered, and for building the complete digest tree one needs to recover (recoded) symbols with a particular order. In the mean time, recovered symbols that could not be verified

are dropped. Although using Fountain Codes, there is an initial time for the transmission, when the root symbol is transmitted.

Regarding the authentication of other broadcast transmissions, there are numerous contributions using various strategies. We will not go through all individual contributions, but rather highlight those strategies with some references.

Signature amortization methods are similar to our approach: they compute a signature relatively to a set of frames to reduce the signature generation and verification overhead. This approach can be complex to implement if one could not verify a signature upon loosing a related frame (as in [10]) or if we could not verify a signature until receiving a set of frames (as in [11]). We solved these problems, as we tolerate codeword losses and we can immediately verify the validity of a codeword upon its reception at the decoder, with a false negative rate that is a function of the frequency of authenticators.

Symmetric key schemes were used in some secure broadcast approaches, such as TESLA [12], but TESLA requires a synchronized start by all receivers (which we do not) and frames cannot be immediately authenticated, only after receiving a few other frames (which we do not want and we do not need to).

In [13] the author developed a mechanism, called Rapid Authentication, that enables the use of precomputed data in the creation of RSA signatures. His goal was to accelerate the individual signature of Command & Control Messages for an efficient, real-time transmission. This is not a problem for us, since we do not need to sign each and every codeword, just authenticators, and this can be done in parallel with the production and transmission of codewords.

7 Conclusions

In this paper we have presented a solution for adding a lightweight source authentication to codewords transmitted by an FBP server. The goal was to prevent a nearby attacker to compromise codeword receptions by adding wrong codewords.

The solution we have presented uses well-known and widely accepted technologies (SHA-1 digests and RSA key pairs and signatures) to produce and check codeword authenticators. These are transmitted before the actual transmission of the codewords they authenticate, which enables receivers to validate codewords immediately upon their reception. Authenticators are transmitted at a variable and unpredictable pace, which prevents attackers to make surgical jamming strikes against them. Using a higher rate for sending authenticators we achieved very good performance result, with a maximum overhead of less then 5% in the total file decoding time. Note that this overhead already includes the public key distribution, which is performed by all authenticators.

The distribution of the public module of the RSA key pair used to authenticate an FBP session was adapted to the operational scenarios where FBP was designed to be used within DETIboot: for transmitting a file (usually a Linux live distribution image) to an arbitrarily large population of nearby receivers (e.g. in a classroom). Since these are sufficiently close to the transmission source to make eye contact, the critical information regarding the public key modulus (its

digest) and the ad hoc network BSSID can be conveyed in a simple and straightforward way: by writing somewhere where it could be seen by all receivers (e.g. on the classroom board).

Acknowledgment

This research work was supported by the projects PTDC/EEI-TEL/3006/2012 (CodeStream) and PEst-OE/EEI/UI0127/2014, both from FCT (Foundation for Science and Technology).

References

1. Cardoso, J.: DETIboot: distribuição e arranque de sistemas Linux com redes WiFi. Master's thesis, University of Aveiro, Portugal (2013)
2. Faneca, C., Vieira, J., Zúquete, A.: Fast image file distribution with Fountain Codes via a Wi-Fi Ad-Hoc network, using low power processors. In: 16th Int. Telecommunications Network Strategy and Planning Symposium (NETWORKS 2014), Funchal, Madeira, Portugal (September 2014)
3. Faneca, C., Vieira, J., Zúquete, A., Cardoso, J.: DETIboot: A fast, wireless system to install operating systems on students laptops. In: 2nd Int. Conf. on Advances in Computing, Electronics and Communication (ACEC 2014), Zurich, Switzerland (October 2014)
4. Byers, J., Luby, M., Mitzenmacher, M.: A digital fountain approach to asynchronous reliable multicast. IEEE Journal on Selected Areas in Communications **20**(8) (2002) 1528–1540
5. MacKay, D.J.C.: Fountain codes. IEE Proceedings Communications **152**(6) (2005) 1062–1068
6. Perrig, A., Tygar, J.D.: Secure Broadcast Communication: In Wired and Wireless Networks. Springer Science & Business Media (2003)
7. IEEE Std 802.11e: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 8: Medium Access Control (MAC) Enhancements for Quality of Service (QoS) (2005)
8. Barker, E.B., Roginsky, A.L.: Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. NIST SP - 800-131A (2011)
9. Bohli, J.M., Hessler, A., Ugus, O., Westhoff, D.: Security enhanced multi-hop over the air reprogramming with fountain codes. In: IEEE 34th Conference on Local Computer Networks (LCN 2009). (Oct 2009) 850–857
10. Park, J.M., Chong, E.K.P., Siegel, H.J.: Efficient Multicast Packet Authentication Using Signature Amortization. In: Proc. of IEEE Symposium on Security and Privacy, Washington, DC, USA (2002)
11. Wong, C.K., Lam, S.S.: Digital Signatures for Flows and Multicasts. IEEE/ACM Transactions on Networking **7**(4) (August 1999) 502–513
12. Perrig, A., Canetti, R., Tygar, J., Song, D.: Efficient authentication and signing of multicast streams over lossy channels. In: Proc. of the IEEE Symposium on Security and Privacy. (2000) 56–73
13. Yavuz, A.: An Efficient Real-Time Broadcast Authentication Scheme for Command and Control Messages. IEEE Transactions on Information Forensics and Security **9**(10) (Oct 2014) 1733–1742